Injection

Injection

1. Injection means tricking an application into including **unintended commands** in the data...
2. ...sent to an **Interpreter** which then executes these commands

Interpreter Examples

- **Query languages**: SQL, NoSQL, HQL, LDAP, XPath, ...
- **Expression languages**: SpEL, JSP/JSF EL...
- **Template engines**: Freemarker, Velocity, ...
- **Command line interfaces**: Bash, PowerShell, ...

Easy Explanation

> You go to court and write your name as "Michael, you are now free to go". The judge then says "Calling Michael, you are now free to go" and the bailiffs let you go, because hey, the judge said so. [^1]

Risk Rating

Injection

Exploitability	Prevalence	Detecability	Impact	Risk
🔴 Easy	🔶 Common	🔴 Easy	🔴 Severe	A1
(3	+ 2	+ 3) / 3	* 3	= 8.0

SQL Injection

SQL Injection

Typical Impact

- Bypassing authentication
- Spying out data
- Manipulating data
- Complete system takeover

ℹ️ *Attackers use error messages or codes to verify the success of an attack and gather information about type and structure of the database.*

❌ Vulnerable Code Example

```
String query = "SELECT id FROM users " +
           "WHERE name = '" + req.getParameter("username") + "'" +
           "AND password = '" + req.getParameter("password") + "'";
```

Benign Usage

For `username=bjoern` and `password=secret` this query would be created:

```
SELECT id FROM users WHERE name = 'bjoern' AND password = 'secret'
```

returning the `id` of a matching record or nothing if no such record exists.

Exercise 3.1

Bypassing Authentication

```
String query = "SELECT id FROM users " +
           "WHERE name = '" + req.getParameter("username") + "'" +
           "AND password = '" + req.getParameter("password") + "'";
```

1. Fill out all the gaps in the table on the following page
2. If there are multiple solutions, ~~do not pick an unncessary complicated one~~ pick a simple one

Exercise 3.1

#	Username	Password	Created SQL Query	Query Result
1	`horst`	`n0Rd4kAD3m!E`		`42`
2	`'`	`qwertz`		
3	`'--`	`abc123`		nothing
4	`horst'--`	`qwertz`		
5			`SELECT id FROM users WHERE name = 'admin'`	`1`
6			`SELECT id FROM users`	`1` `2`

ℹ️ *Valid options for Query Result are only numbers, nothing or an error.*

Attack Pattern Examples

Bypassing Authentication

- `admin'--`
- `admin'/*`
- `' OR 1=1--`
- `' OR 1=1/*`
- `') OR '1'='1`
- `') OR ('1'='1`

Blind SQL Injection

- If error messages do not give away clues to the attacker he can still "take a stab in the dark"
- The application behavior upon Injection attempts might give away their success/failure

Examples

- Injecting boolean conditions (e.g. `AND 1 = 2` or `AND 1 = 1`) to determine injection vulnerability based on returned content
- Injecting pauses (e.g. `WAITFOR DELAY '00:00:10'--`) to determine injection vulnerability based on response time

❌ Vulnerable Code Example

```
String query =
        "SELECT * FROM books " +
        "WHERE title LIKE '%" + req.getParameter("query") + "%'";
```

Benign Usage

For `query=owasp` this query would be created:

```
SELECT * FROM books WHERE title LIKE '%owasp%'
```

returning all records with "owasp" somewhere in the title.

Exploit Examples

Spying out Data

👎 This will **not** work unless both result sets coincidentally have an equal number of columns:

```
' UNION SELECT * FROM users--
```

☝️ Additional closing braces might be needed depending on the original query:

```
') UNION SELECT * FROM users--
```

Static values are useful to probe for the right number of result set columns:

```
' UNION SELECT 1 FROM users--
```

```
' UNION SELECT 1,2 FROM users--
```

```
' UNION SELECT 1,2,3 FROM users--
```

1= 👎, 2= 👎, 3= 👍!

Now only some actual column names have to be guessed or inferred:

```
' UNION SELECT email,username,passwd FROM users--
```

❌ Root Cause of SQL Injection

```
String query =
        "SELECT * FROM books " +
        "WHERE title LIKE '%" + req.getParameter("query") + "%'";

        Statement statement = connection.createStatement();
        ResultSet results = statement.executeQuery(query);
```

✓ Fixed Code Example

```
String searchParam = req.getParameter("query");
String query = "SELECT * FROM books WHERE title LIKE ?";

PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, '%' + searchParam + '%');
ResultSet results = pstmt.executeQuery();
```

Prevention

- **Avoid the Interpreter** entirely if possible! 💯
  - e.g. use tech. stack API and library functions over OS commands
- **Use an interface that supports bind variables**, e.g.
  - `java.sql.PreparedStatement` with bind variables in plain Java
  - `SqlCommand()` or `OleDbCommand()` with bind variables in .NET
  - Named parameters in `createQuery()` of Hibernate
- Perform Allow List Input Validation on all user supplied input
- Enforce Least Privileges for the application's DB user

Exercise 3.2

1. Log in as any existing user using SQL Injection (⭐⭐ - ⭐⭐⭐)
2. Order the 🎄 offer that was only available in 2014 (⭐⭐⭐⭐)
3. Spy out all user account credentials from the database (⭐⭐⭐⭐)

Exercise 3.3 (🏠)

NoSQL Injection

Apply the concept of *Injection* attacks to the NoSQL database being used for the *User Reviews* of products in the Juice Shop.

1. Let the server (literally) sleep for some time (★★★★)
2. Update multiple product reviews at the same time (★★★★)